

**METHOD AND APPARATUS FOR DEBUGGING
PROGRAMS IN A DISTRIBUTED ENVIRONMENT**

5

Priority

This application claims priority to U.S. provisional patent application Serial No. 60/189,521 filed March 15, 2000 and entitled "Method and Apparatus for Debugging Heterogeneous Processors."

Copyright

10

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

15

Background of the Invention

1. Field of the Invention

The present invention relates to the field of debugging programs in a distributed environment, such as a set of heterogeneous hardware processors (integrated circuits or In-Circuit Emulators), and/or software-based simulators.

20

2. Description of Related Technology

25

The process of debugging software intended for operation in embedded applications is a task that employs many different tools depending on the stage of software readiness. Programs are often organized in a hierarchical fashion, but need not include such structure. While structured programs are easier to debug because there is a reduced span of control within the software, both structured and unstructured programs commonly employ similar programming techniques including functions, subroutines, co-routines, and the like. Software-based simulators are used to provide design engineers and programmers (hereinafter "engineers") with absolute control over the execution of the software to be debugged. This process permits control at varying degrees of granularity ranging from a single line of code to larger blocks such as procedures, functions, and subroutines. Control includes among other facilities, the ability to start and stop execution, report results and change memory contents.

30

Multiprocessor systems complicate the debugging process significantly as compared to a unitary processor hardware environment. The most complex debugging environment is one in which the processors used in the multi-processor hardware employ different instructions sets. This condition is known to those skilled in the art as a heterogeneous multiprocessor system.

5 Heretofore, engineers have relied on diverse development environments, often provided by many different vendors, to debug such heterogeneous systems. These diverse development environments often provide different user interfaces, different commands, different capabilities, and sometimes employ different underlying computer operating systems which makes information transference between the systems challenging at best.

10 These conditions force engineers involved with debug to be less productive than would otherwise be possible were the interfaces common between all of the different processors. Further, each different system requires engineers to become conversant with its unique capabilities and disabilities, which requires more learning time and results in reduced productivity.

15 Additionally, the use of such heterogeneous development environments potentially introduces more error, due largely to the fact that the development environment associated with one of the heterogeneous processors utilizes one function for a given keystroke or other user input, while a different environment may utilize an all together different function for that same keystroke or input. Accordingly, the engineer must in effect maintain a "correlation table" for
20 the various functions and associated inputs depending on which development environment is being used.

Based on the foregoing, what is needed is an improved method and apparatus for debugging devices such as heterogeneous processors. Such method and apparatus would ideally be readily adaptable to a number of different hardware/software environments, would allow for ready transfer
25 of information associated with one processor to the development/debug environment of another, thereby facilitating side-by-side comparison of the operation of the different processors. Such improved debug method and apparatus would also be readily adapted to run on conventional microprocessor-based platforms, and accommodate inputs from both the aforementioned hardware processes and simulation processes.

Summary of the Invention

The present invention satisfies the aforementioned needs by an improved method and apparatus for debugging devices such as heterogeneous processors.

In a first aspect of the invention, an improved method for debugging programs in a distributed environment is disclosed. In one exemplary embodiment the environment comprises heterogeneous hardware digital processors (integrated circuits or In-Circuit Emulators), and/or software-based simulators, and the method comprises: identifying a plurality of processes; initializing each of the processes; executing with a single thread of control among the processes; and continuously cycling among the processes to obtain status information. Each "running" simulation process simulates the execution of a single instruction for each status request. In a second exemplary embodiment, the method further comprises initializing profile information, and incrementing the profile history after simulation (simulator) or execution (hardware) of one instruction.

In a second aspect of the invention, an improved computer program useful for debugging such distributed programs is disclosed. In one exemplary embodiment, the computer program comprises a C++ source code listing reduced to an object code representation and stored on the magnetic storage device readable by a microcomputer, and adapted to run on the central processing unit thereof. The computer program further comprises an interactive, menu-driven graphical user interface (GUI), thereby facilitating ease of use. The basic design of the computer program takes an object-oriented approach, with an abstract class defined to provide the interface to an individual process within the target system.

In a third aspect of the invention, an improved debug architecture is disclosed. In one exemplary embodiment, the improved debug architecture comprises a digital processor with a debug process running thereon, at least one simulation process associated and in data communication therewith, and at least one hardware process in data communication with the processor, wherein the simulation and hardware processes are executed with a single thread of control via the debug process.

In a fourth aspect of the invention, an improved apparatus for running the aforementioned debug computer program is disclosed. In one exemplary embodiment, the system comprises a stand-alone microcomputer system (e.g., IBM PC) having a display, central processing unit, data

storage device(s), and input device. The apparatus is adapted to run one or more of the
aforementioned simulators and the debug program, and interface with one or more hardware
processes external to the apparatus via respective data interfaces. The engineer may then debug
the multiple hardware processes using the debug program and simulation process(es),
5 advantageously avoiding the need for multiple hardware/software environments for the debug
and simulation processes.

Brief Description of the Drawings

10 Fig. 1 is a logical flow diagram illustrating one exemplary embodiment of the general
debugging methodology employed the present invention. OBJ
←

Fig. 1a is logical flow diagram illustrating an alternate embodiment of determining the
sleep interval based on poll delay according to the invention.

Fig. 2 is a logical flow diagram illustrating a second embodiment of the debugging
15 methodology employed in the present invention, incorporating profile histories.

Fig. 2a is a logical flow diagram illustrating the use of the method of Fig. 2 to optimize
the systems performance of a multi-processor based system.

Fig. 3 is a block diagram of an exemplary multi-processor debugging architecture for
which the methodology of Fig. 1 may be applied.

20 Fig. 4 is a functional block diagram of one exemplary embodiment of a computer system
useful for running a computer program embodying the method of Fig. 1.

Detailed Description

Reference is now made to the drawings wherein like numerals refer to like parts
25 throughout.

As used herein, the term "processor" is meant to include any integrated circuit or other
electronic device capable of performing an operation on at least one instruction word including,
without limitation, reduced instruction-set core (RISC) processors such as the ARCTM user-
configurable core manufactured by the Assignee hereof, central processing units (CPUs), and
30 digital signal processors (DSPs). The hardware of such devices may be integrated onto a single

piece of silicon ("die"), or distributed among two or more dies. Furthermore, various functional aspects of the processor may be implemented solely as software or firmware associated with the processor.

As used herein, the term "process" refers to executable software that runs within a processor environment. This means that the process is typically scheduled to run based on a time schedule or system event. It will generally have its own Process Control Block (PCB) that describes it. The PCB may include items such as the call stack location, code location, scheduling priority, etc. The terms "task" and "process" are often interchangeable with regard to computer programs.

Similarly, a "task" as used herein generally refers to a process-like entity whose PCB is referred to as a Task Control Block (TCB). A "thread" refers to a process having the same properties as a task except that it runs within a task context and uses the task's TCB. Multiple threads can run within the context of a single task. Threads are often more efficient than tasks because they don't require as much time to be switched into CPU context when the task they are associated with is already running.

Overview

In general, the present invention provides a flexible system for debugging programs in a distributed environment. In one embodiment, such a distributed environment comprises a set of heterogeneous hardware processors (integrated circuits or In-Circuit Emulators), and/or software-based simulators; see the discussion with respect to Fig. 3 below.

The basic design takes an object-oriented approach with an abstract class defined to provide the basic interface to an individual process within the target system. Among other benefits, the use of an object-oriented approach allows language-independence at the design level. Object-oriented programming is well understood in the computer programming arts, and accordingly will not be described further herein.

The methods and instance variables of this abstract class fall into two categories: (i) those relating to direct control of the target processor and examination of its state; and (ii) those relating to synchronization and control of the individual processes by the debugging system.

Instance variables relating to direct control of the target processor include, *inter alia*, setting and examining register values and the contents of memory; starting, stopping, and "single-stepping" the processor; setting hardware-controlled breakpoints; and similar operations. Registers are often employed by engineers (or the output of compilers) to hold working variables; i.e., those that are being operated upon. These registers frequently contain values of substantial interest to the design engineer/programmer. Consequently, it is important for the debug environment to provide access to such register values. Many programming errors may be temporarily corrected by a skilled engineer through affording the ability to modify the contents of these registers. Such is often the case with programmatic loop constructs that either terminate prematurely or fail to terminate at the intended time. By modifying a register value, the engineer performing the debugging task may be able to permit the program to proceed further in the program sequence while producing correct results. This ability significantly reduces development time by permitting the engineer to make changes that are local in scope, yet have a global impact on the functioning of the software.

So-called "single stepping" of a processor, whether as a simulation process or a hardware process, permits engineers to follow the execution of the software to determine where implementation or design errors exist. Generally accepted debugging practice is defined, *inter alia*, in IEEE 1008-1987 IEEE Standard for Software Unit Testing and terms in IEEE Std 610.12-1990, Standard Glossary of Software Engineering Terminology (ANSI).

Synchronization and control variables include, *inter alia*, time of last status check; the current delay between status checks; whether the process is running in a simulator or in hardware; the processor type and options; and similar parameters.

In the present embodiment of the invention, individual subclasses are defined for each supported processor, In-Circuit Emulator (ICE), or simulator with each subclass being implemented generally as a dynamically loadable library. By creating individual subclasses for each processor, it is possible to dynamically link processors into the system without the need to recompile or otherwise reconfigure the static structure of the debug environment. The use of a dynamically loadable library permits changes to be made while the debugger is in operation without necessarily halting the execution of other processors. Further, this permits the engineer

performing debug activities to simulate the failure of one or more processors by selectively switching them out of the debug environment. In addition, it is possible to substitute the operation of actual hardware in place of a simulator or in-circuit emulator to permit continuous debugging as more information is gathered.

5 Some digital processor families reserve a section of the processor instruction set for so-called "extended operations" or "extensions" which are typically implemented in customized sections of the hardware to perform application-specific functions such as Viterbi decode, FFT, and the like. To handle these extended operations in the debugger, the processor instance class for those processor types defines a further interface to dynamically loadable libraries which
10 embody one or more of the possible extended operations. When the instance is an interface to a hardware processor, the libraries provide the functions for displaying the extension instruction in machine code listings.

When the instance is a software simulator, the extension library must also provide the implementation of the instruction itself. It will be appreciated that software simulators operate
15 by implementing the logical operation of hardware in software. Hardware functions are performed by sequences of software instructions in the simulator. State information and registers are assigned specific memory locations in the simulation software memory space. Within the ARC design environment, the hardware extension library contains the HDL model for the hardware used during the compilation phase. Since the extension may also be implemented
20 during the debug phase by a software simulator, it is necessary to also provide this software in the extension library.

When debugging, each process will normally be in one of two states. It may be stopped, with execution suspended while the user examines and possibly modifies the process state before continuing. Alternatively, the process may be running, in which case it executes instructions
25 until it reaches a breakpoint, error condition, a certain amount of time has passed, or the user manually requests the operation be suspended. It will be recognized that the foregoing list of conditions under which a running process is terminated is not all inclusive; other situations where the process terminates may exist.

When running, it is desirable for the process to execute, as nearly as possible, at the speed with which execution would occur if it were not running under the control of a debugger. In this way, the actual operation of the process on the device is most closely simulated. However, it is also desirable to rapidly and continuously update the status displays for the user. In practice, obtaining status information from hardware processors often takes a large amount of time as compared to the execution of a single instruction; so the use of continuous status requests would significantly impair performance. Conversely, when running in a simulator, obtaining the status has relatively little impact; some care must be taken, however, to keep multiple simulations synchronized as though they were running on parallel hardware processors. Furthermore, it is desirable that the debugger itself run as efficiently as possible to reduce the impact on any other programs which may be concurrently executing.

To address the foregoing issues, the debugger of the present invention executes with a single thread of control which, when in "run" mode, continuously cycles among the various processes obtaining status information. Each "running" simulation process will simulate the execution of a single instruction for each status request. Associated with each running hardware process is an indication of when the status was last checked, and a variable delay interval indicating when it should next be checked. This association occurs as a consequence of employing an object oriented language such as C++, but may also be implemented by use of explicit parameters passed via function or subroutine calls. Alternatively, the association may be made completely explicit by use of a common data structure such as mailboxes, message buffers and similar communications protocols commonly used by operating systems to manage such data. In the instance where all such running processes are executing on hardware processors, then each iteration through the status loop further includes an idle period designed to delay the debugger until at least one process is ready to be checked.

25

Methodology

Referring now to Fig. 1, the generalized method of debugging distributed programs according to the present invention is described. The exemplary embodiment of Fig. 1 is described in terms of a computer program, although it will be recognized that such program is

NOT TAUSCET
4

only one means for implementing the method of the invention. For example, certain portions of the functionality described herein could be implemented in hardware if desired.

As illustrated in Fig. 1, the method 100 generally comprises the steps necessary to repeatedly obtain the status of each running processor in sequence, subject to the aforementioned conditions. If any of the running processors are being simulated, the simulation of the illustrated embodiment will be advanced by one instruction before its status is checked. To allow processes running on hardware to run as efficiently as possible, they will only be checked when a specified per-process time interval has elapsed since the previous check. This time interval may be varied (either statically, such as by merely changing the duration of the interval, or dynamically, such as based on the output of an associated algorithm which calculates a time interval based on other parameters), and is set according to the attributes of the particular process and hardware. At the end of each check cycle, if it is determined that all running processes are executing on hardware, and none of the running processes are ready to be checked, a predetermined delay may be invoked to prevent unnecessary passes through the check loop. Similar to the per-process interval period, the predetermined delay may be varied (statically or dynamically) as well.

In step 102, the poll delay associated with each process is initialized. As used herein, the term "poll delay" refers to the minimum time period between retrievals of processor status information for display to the user. In the present embodiment, the poll delay is initialized to the minimum value, and its next poll time set to "now". As used herein, the term "now" is used to indicate the current point in time at which "now" is referenced. Next, in step 104, the "ran simulator" value is set to "false", and the "need sleep" value also set "false". The "ran simulator" and "need sleep" values determine the desirability of introducing a delay before the next cycle of processor status checks.

In step 104, the debugger state is determined; if in "run", the program proceeds to step 106, where for each process, the run status of that process is determined (step 108). If the debugger is in a state other than run (e.g., stop), then the process returns as shown in Fig. 1. For each process, if the process is running, the process type is next determined in step 110. The term "process type" as used herein refers to whether the processor is running in the simulation or hardware environment.

If the process type is the simulation environment, the program proceeds to step 112, where the simulated processor is advanced through one instruction cycle. The simulated processor's status is subsequently checked. The "ran simulator" value is then set to "true", and the program returns to step 108 again for the next process.

5 In the hardware environment (step 110), the value of the "next poll" is determined per step 114; if it represents a time in the future relative to the present time, the "need sleep" value is set to "true" per step 116, and the program returns to step 108 again. This in effect delays the program for a predetermined time until the next polling opportunity is available, as previously described. If the value of the next poll is the current time or a time in the past, the polling
10 opportunity is immediately available, and the status of the process is checked per step 120. The value of "next poll" is set to the current time plus the processor's poll delay per step 120 as well. The "check time" value is also set to the minimum of the current "check time" or the "next poll" value in this step 120 as well. Note that the status check of step 120 may change the poll delay value to better balance the need to display processor status with the desire to minimize the use of
15 system resources. This need is determined by monitoring the current systems resource availability and comparing that to the amount that may be required to perform both the display function and other functions that may be operating concurrently. This monitoring is well known to those skilled in the art of operating systems design as part of "load balancing." Proper load balance may be determined by a number of techniques known to those of ordinary skill in the art of
20 operating systems design, the specific techniques of which are not relevant to the present invention. For example, such techniques are taught in "Operating Systems Principles" by Per Brinch Hansen, Prentice Hall 1973. After these operations have been completed, the program returns to step 108.

After the foregoing steps 110 through 120 have been completed for each process, the
25 program checks the value of the "ran simulator" variable per step 122 to determine whether any of the running processes are executing in simulators. If so, the program advantageously returns immediately to step 104 so that the simulation(s) will run as quickly as possible, as is desired in order to most closely replicate the actual operating conditions of the simulated device. If no running processes are executing in simulators, then the program proceeds to step 124 to
30 determine if any of the hardware processes were not yet ready to be checked, as indicated by the

"need sleep" value set to "true". If all hardware processes have been checked, the program returns to step 104. If at least one hardware process was not ready to be checked, the program sleeps until the next "check time" per step 126. At the next check time, the program awakes and returns to step 104.

5 It will be recognized that while the aforementioned poll delay and "need sleep" intervals are described in terms of predetermined, fixed time periods, these intervals may alternatively be variable in nature, depending on the value of other parameters or the existence of other circumstances within the hardware/simulation environments. For example, in one alternate embodiment, the value associated with the "need sleep" interval is algorithmically determined
10 based on analysis of the value of the "next poll" determination in step 114. Specifically, as illustrated in Fig. 1a, the current time (referenced to the time when step 114 is executed) is subtracted from the value of the "next poll" determined in step 114 to arrive at a minimum delay value before the polling of that device is available. In this fashion, the "need sleep" interval is dynamically adjusted based on the next available polling opportunity, thereby reducing an
15 "extra" delay introduced by a fixed sleep interval.

 In another embodiment, the required sleep interval is determined based on statistical analysis of historical data obtained either from past debug operations for the hardware environment under analysis, from operating history generated immediately prior to the poll delay determination in step 114 (such as using a moving "window" technique of the type well known
20 in the art), or some combination thereof. Numerous types of statistical/historical analyses and associated algorithms are known to those of ordinary skill in the programming arts, and accordingly are not described further herein.

 Referring now to Fig. 2, an alternate embodiment of the method of Figs. 1-1a is described. Optimizing systems performance in a single processor system is a relatively simple
25 matter of moving applications code to better utilize the CPU, modifying the code to better reflect the systems capabilities, and/or developing a new algorithm for the specific circumstances under which the system is operating. For multiple processor systems, however, this process becomes much more complicated. Each individual processor can be optimized for locally optimum performance but the systems may still have suboptimal performance. The methodology 200 of
30 Fig. 2 advantageously permits engineers to perform both local and global optimizations across

multiple processors based on the execution history of the system when data representative of "real world" inputs is supplied. Alternatively, real world data may be supplied to the system for the purpose of collecting execution history profiles. These profiles may identify: (i) individual instructions of a program running on a specific processor; (ii) sequential blocks of code running on a specific processor, and (iii) functions, subroutines or other information determined by the engineer to be valuable during the optimization process. Such information may include, for example, patterns of register references, number of memory references, patterns of memory reference, and the like.

In the modified run loop of Fig. 2, the aforementioned profile history is initialized during the beginning of the run loop (step 203), and profile history collected (steps 213, 215) after both the simulations step and hardware execution step. These two profile history data collection functions simply record execution information specified by the engineer for later presentation. The modified run loop of Fig. 2 operates in the same manner as that of Figs. 1-1a in all other respects.

Referring now to Fig. 2a, the method 200 of Fig. 2 is employed as a step in the method 250 of optimizing the performance of a mutliprocessor system. The run loop function of Fig. 2 is performed to gather execution history as may be determined by the user's selection of information to be gathered (step 252). These results are then examined (step 254) to determine which if any processors are relatively overloaded, and which if any are relatively underloaded. Portions of executable code may be rearranged within the scope of a single processor's program space, restructured so as to consume fewer resources, or partitioned across processors so as to more economically computer the desired result (step 256). The aforementioned improvements to the performance of the multi-processor based system are well known to one of ordinary skill in the art of programming multi-processor systems and is not further discussed here. Once the improvements are made, the run loop function is again executed (step 258) to gather execution history data that will prove or disprove the performance. The process continues iterating until the performance goals determined by the user have been met, or the user has determined that the goals can not be met.

It will be recognized that while the foregoing example and description with respect to Figs. 1, 1a, 2, and 2a herein is cast in terms of a specific series of steps for accomplishing the

desired result (i.e., debugging in a distributed environment), various permutations of this series of steps, including substitution and/or addition of other steps, may be used consistent with the invention disclosed herein. Accordingly, the scope of the disclosed invention should be determined by the claims appended hereto, without respect to specific embodiments or
5 limitations presented within the foregoing discussion.

Referring now to Fig. 3, one exemplary multiprocessor debugging architecture 300 for which the foregoing method may be used is described. The architecture 300 generally comprises a debugger/simulator process 302, and a plurality of hardware processes 304 each operatively coupled via respective data paths 306 and control paths 308 to the debugger/simulator 302. Note
10 that the control paths 308 are unidirectional, whereas data may flow both to and from the hardware processes 304 via the data paths 306. It will be recognized that, depending on the type of hardware platform employed to implement the debugger/simulator 302 (see Fig. 4 below for one exemplary embodiment), various types of data and control pathway hardware may be employed, such as RS-232, IEEE-1394 "Firewire", or even fiber or wireless links, so long as any
15 required timing relationships are preserved. Alternatively, the debugger/simulator 302, including data and control paths, may be physically integrated with the hardware processes, such as by being disposed entirely within a single silicon substrate. For example, the debugger/simulator may be employed as an algorithm running on a RISC processor, CISC microprocessor, digital signal processor (DSP), or other digital processor associated with the individual hardware
20 processes.

The debugger/simulator 302 of Fig. 3 comprises a plurality of individual simulator processes 310, operatively coupled to a debugger process 312 via additional respective data paths 314 and control paths 316. The debugger and simulator processes 310, 312 in one embodiment comprise software implementing the foregoing methodology, although it can be appreciated that
25 at least portions of the methods of Figs. 1 and 2 may be embodied in firmware or even hardware if desired.

Each simulator process 310 (1 through N) of Fig. 3 is representative of a single instance of a simulator that functions as the target central processing unit (CPU) of the complete system. Each simulator process 310 may implement any instruction set architecture as is needed by the
30 actual designed heterogeneous multiprocessor system. Simulation processes may be used for a

number of reasons including unavailability of hardware, a desire to control systems debug such that hardware transient behaviors are not present, and for reasons of cost. Likewise, each hardware process 304 represents a single instruction set architecture within the heterogeneous multiprocessor system. These processes may be actual physical semiconductor devices, or an in-circuit emulator (ICE) of the type well known in the art. Hardware processes may be used when fast execution is desired, debugging of hardware interfaces is taking place, or the actual operation of the hardware is in some way different than that exhibited by the simulation model.

Each of these processes, either hardware or software, may implement any desired instruction set architecture or a fixed function operation. Examples of an instruction set architecture include but is not limited to: Intel 8080, 8086, 80386, Pentium, Motorola 68000, 68030, PowerPC, Texas Instruments TMX320C6100 and the like. Fixed function operations may include, but it not limited to: special purpose hardware such as Viterbi decode, digital filters, noise shapers, FFT, and the like.

Accordingly, the present invention is advantageously compatible with systems represented by only simulator processes or hardware processes, as well as those that are represented by a combination of simulation and hardware processes. These processes may be homogeneous or heterogeneous in nature, thereby providing the engineer with additional flexibility not present in prior art techniques.

Apparatus for Implementing Methodology

Referring now to Fig. 4, one embodiment of a computing device capable of implementing the debugging methods (in the form of a computer program) discussed previously herein with respect to Figs. 1-2a is described. It is noted that the foregoing methods are readily reduced to source code listings in any useful higher level programming language, such as for example C++, and subsequently compiled, by one of ordinary skill in the computer programming arts. Appendix I hereto provides one such exemplary source code listing.

The computing device 400 comprises a motherboard 401 having a central processing unit (CPU) 402, random access memory (RAM) 404, and memory controller 405. A storage device 406 (such as a hard disk drive or CD-ROM), input device 407 (such as a keyboard or mouse), and display device 408 (such as a CRT, plasma, or TFT display), as well as buses necessary to

support the operation of the host and peripheral components, are also provided. The method of Fig. 1 are embodied in the form of an object code representation of a computer program and stored in the RAM 404 and/or storage device 406 for use by the CPU 402 during analysis, the latter being well known in the computing arts. Alternatively, the computer program may reside on a removable storage device (not shown) such as a floppy disk or magnetic data cartridge of the type also well known in the art. The user (not shown) analyzes the data input from the various sources (such as heterogeneous processors) by inputting initiating operation of the computer program via the program displays and the input device 407 during system operation. Alternatively, the system may be configured to automatically accept (and store if desired) the various data inputs and run the computer program when sufficient data exist, or on a periodic or ongoing basis. Many such alternative are possible, each being well within the skill of the ordinary practitioner. Analyses and/or formatted data generated by the program are stored in the storage device 406 for later retrieval, displayed on the graphic display device 408 for viewing by the user, or output to an external device such as a printer, data storage unit, other peripheral component via a serial or parallel port 412 if desired.

It may be appreciated that any number of types of information may be displayed on the graphic display device. The following is illustrative and not prescriptive for such data information. The actual program code executed by the process may be displayed in source code format, assembly language format, or a numerical radix based format where the radix corresponds to a word size or other informative division of data. Execution traces of addresses, address ranges, data values, subroutine entry/exit and the like may also be displayed. For some debug environments interprocessor communications and internal program or processor state information may be displayed. The foregoing information may be displayed in a variety of forms convenient to the user such as bar graphs, histograms, "eye charts", flowcharts, or textual forms.

One preferred embodiment of the hardware used in conjunction with the debugger program previously described herein is based on the industry standard IBM Personal Computer architecture operating on an Intel microprocessor. Such a computer generally comprises a display mechanism such as a CRT display, input devices such as a keyboard and mouse, storage media such as a hard disk drive, communications ports to communicate with any external hardware such as target hardware boards or in-circuit emulators (ICEs). Other alternatives include

workstations manufactured by Sun Microsystems of Mountain View California based on the Sun SPARC microprocessor. These workstations employ peripherals such as those listed above for the IBM Personal Computer but operate internally on a different microprocessor, systems bus, and UNIX-based operating system. However, these examples are merely illustrative, and not
5 prescriptive of the type of hardware on which the invention can operate. In addition to the so-called "clone" machines of the above named companies, there are many other alternative personal computers and workstations upon which the invention can operate. These include those manufactured by Hewlett-Packard, Intergraph, Data General, Apple Computer, and others.

While the above detailed description has shown, described, and pointed out novel features
10 of the invention as applied to various embodiments, it will be understood that various omissions, substitutions, and changes in the form and details of the device or process illustrated may be made by those skilled in the art without departing from the invention. The foregoing description is of the best mode presently contemplated of carrying out the invention. This description is in no way meant to be limiting, but rather should be taken as illustrative of the general principles of
15 the invention. The scope of the invention should be determined with reference to the claims.

APPENDIX I

Copyright © 2000-2001 Metaware Software, Inc. All rights reserved.

```
5 NOINLINE void check_for_completion(  
    void each()->(common_RCE*!),    // Set of command processors.  
    common_RCE *only_this_one, bool &running,  
    bool &repeat_check_immediately,  
    unsigned *sleep_timep=0) {  
10     if (sleep_timep) *sleep_timep = 0;  
    // Polling. If all the running processes are non-simulators, should  
    // sleep in-between polling. Otherwise, can run the simulators  
    // and check the real machine in between, as long as wait long enough  
    // so as not to burden the real machine. i.e., the loop is:  
    // repeat ...  
15     // for each running R do  
    //     if R is a simulator, check its status.  
    //     else if R is hardware, if we've waited  
    //     long enough since last check, check it  
    //     else early = min(early, R's last check time + min wait)  
20     // end for  
    // if there were no simulators, sleep (current_time-early)  
    // end repeat  
    //  
25     // With the GUI this process will be different. The GUI will issue  
    // status checks periodically.  
  
    running = FALSE;  
    repeat_check_immediately = FALSE;  
    bool ran_a_simulator = FALSE, need_sleep = FALSE;  
30  
    unsigned earliest_we_can_check_hardware = (unsigned)-1;  
    unsigned now() { return get_system().get_llio().walltime_milliseconds();  
}  
35     bool changed_status = FALSE, somebody_wants_checking_always = FALSE;  
    changed_status = FALSE;  
    somebody_wants_checking_always = FALSE;  
    for R <- each() do {  
        if (only_this_one != 0 && only_this_one != R) continue;  
        dbg && printf("check for completion on %s\n",  
40             R.get_system().get_name());  
        if (R.get_process() == 0) continue;  
        dbg && printf("checking process [%d]:\n", R.rce_number);  
        dbg && (R.display_processes(), 0);  
        bool was_executing = R.get_process().is_executing();  
45        void check_status() {  
            if (!R.get_system().is_simulator())  
                dbg && printf("!!status check on HW [%d] %s\n", R.rce_number,  
                    R.get_system().name());  
            R.status(0, FALSE, !was_executing);  
50        }  
        System *R_system = R.get_system();  
        0 && printf("system %s wase %d csa %d\n",  
            R_system.name(), was_executing, R_system.check_status_always);  
        if (R_system.check_status_always)  
55            somebody_wants_checking_always = TRUE;  
        if (was_executing || R_system.check_status_always) {  
            // Warning: each time you do a status check, process  
            // could have terminated, so verify that process is non-zero.  
            if (R_system.is_simulator()) {
```

```

        check_status();
        ran_a_simulator = was_executing
        // Either was executing or is now executing.
        || R.get_process() && R.get_process().is_executing();
5    }
    else {
        unsigned Now = now();
        unsigned POLL_WAIT = R.get_process().get_key().delay.
            get_poll_delay();
10    if (FALSE && globals.trace_sleep) {
        printf("[%d]POLL WAIT comes back as %d ",
            R.get_rce_number,POLL_WAIT);
        printf("delta is %d ",Now - R.time_since_last_poll);
        printf("earliest %d\n",earliest_we_can_check_hardware);
15    }
        if (Now - R.time_since_last_poll >= POLL_WAIT) {
        // printf("polling [%d]\n",R.get_rce_number());
        check_status();
        Now = now(); // Status might have taken a while.
20    R.time_since_last_poll = Now;
        // Now get the next value of delay, which may have
        // gone down (e.g., hostlink) or up. This has
        // the necessary side effect of changing the delay.
        if (R.get_process())
25
            POLL_WAIT =
            R.get_process().get_key().delay.current_poll_delay();
        }
        else need_sleep = TRUE;
30    unsigned earliest = R.time_since_last_poll + POLL_WAIT;
        earliest_we_can_check_hardware = _min(
            earliest_we_can_check_hardware,earliest);
        }
        bool is_executing =
35    // The process may have died as a result of checking status.
        R.get_process() && R.get_process().is_executing();
        running = running || is_executing;
        changed_status |= was_executing != is_executing;
        }
40    Process *process = R.get_process();
        dbg && process && printf("!pic %d cd %d\n",
            process.is_executing(), R.completion_delayed);
        // The process may have died as a result of checking status.
        if (process && !process.is_executing()) {
45    if (R.completion_delayed) {
        // Announce status.
        dbg && printf("Call step completion\n");
        R.step_completion(R.completing_stmt_step,TRUE);
        }
50
        process.get_key().delay.minimize_poll_delay();
        }
        //
        repeat_check_immediately = TRUE;
55    }
    if (!ran_a_simulator && need_sleep && running) {
        unsigned Now = now();
        if (Now > earliest_we_can_check_hardware);
        else {
60    unsigned sleep_time = earliest_we_can_check_hardware-Now;
        if (sleep_time) {

```

```

        // printf("!not sleeping for %d; let GUI do
it.\n",sleep_time);
        *sleep_timep = sleep_time;
    }
5      else {
        get_system().get_llio().sleep(sleep_time);
        // printf("!sleep for %d\n",sleep_time);
        }
10     }
    }

```

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
223